2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Evolving Decision-Making Functions in an Autonomous Robotic Exploration Strategy using Grammatical Evolution

Mohd Faisal Ibrahim[1] and Bradley James Alexander[2]

Abstract— **Customising navigational control for autonomous robotic mapping platforms is still a challenging task. Control software must simultaneously maximise the area explored whilst maintaining safety and working within the constraints of the platform. Scoring functions to assess navigational options are typically written by hand and manually refined. As navigational tasks become more complex this manual approach is unlikely to yield the best results. In this paper we explore the automatic derivation of a scoring function for a ground based exploration platform. We show that it is possible to derive the entire structure of a scoring function and that allowing structure to evolve yields significant performance advantages over the evolution of embedded constants alone.**

I. INTRODUCTION

Setting up navigational control for robotic platforms for autonomous mapping in unknown environments remains a challenging task. Control software must maximise area explored whilst maintaining safety and working within the kinematic and power constraints of the robotic platform. Lack of a-priori knowledge of the exploration environment obliges the control software to make navigational decisions dynamically on the basis of provisional information. These navigational decisions must come from an implicit or explicit ranking of immediate navigational options [1]. Where the ranking is explicit, some form of scoring function must assign a level of utility to each navigational option. Subsequently the navigational option with the highest utility is selected as the next navigational goal. The exact form taken by this scoring function depends on many factors including: the nature of the environment; the robot hardware and sensors; computer processing resources; the algorithms used to extract signals from the environment; and the way in which navigational options are specified. Unsurprisingly, given this diversity of context, there is a corresponding diversity of approaches to scoring navigational options [1], [2], [3], [4], [5], each of which is well-suited to its particular context.

One common characteristic of most current approaches is that the scoring function they use to rank navigational options is composed by hand. Given the complexities of platform, sensors, software, operating environment, and navigational tasks it is highly unlikely that these hand-written scoring functions were derived – perfectly formed – from first principles [6]. Much more likely, they are at least partly the product of experimental refinement. In other words, we assert that handwritten scoring functions are, in part, the product of a *search process*. As the complexity of navigational environments grow and the number of factors to consider increases, the effectiveness of hand-driven search for a scoring function will become harder to sustain. This being so, an interesting question is the extent to which this search process can be automated and what benefit can be derived from such automation?

[1]M. F. Ibrahim is with Department of Computer Science, The University of Adelaide, SA 5005, Australia and Universiti Kebangsaan Malaysia `mohd.f.ibrahim at cs.adelaide.edu.au`

[2]B. J. Alexander is with Department of Computer Science, The University of Adelaide, SA 5005, Australia `brad at cs.adelaide.edu.au`

A. Contributions

In earlier works [7], [8], we demonstrated that evolutionary search can be used to successfully evolve numeric constants in a scoring function and show, in simulation, that the evolved functions can outperform handwritten code. In this work we extend the function search space substantially to include the *structure* of the scoring function. We show, in simulation, that allowing constants, functions and structure of the scoring function to evolve produces better navigational outcomes than evolving constants alone. The ability of evolve structure is a significant step as it eliminates bias arising from predetermined structure, thus giving the control function much more freedom to adapt to its context.

In our experiments, the evolved solutions generalise well to a more challenging environment. Moreover, we have found, by sampling evolved scoring functions, that structurally different scoring functions can embed very similar ranking relationships. These relationships appear to reveal the important trade-offs between exploration, safety and power usage in the given framework.

The remainder of this paper is organised as follows. In the next section we describe the application domain and related work. In section III we describe the exploration strategy used by the robot. In section IV we outline our evolutionary framework. In section V we describe our experimental setup. Our results are presented and discussed in section VI. Finally, we conclude in section VII.

II. APPLICATION DOMAIN AND RELATED WORK

This work applies to autonomous exploration of unknown environments by an unmanned ground vehicle (UGV's). We assume that navigation works in a control-loop where a new navigational goal is chosen, either, after a previous goal is reached or a predetermined time has elapsed. We assume that the body of the control loop has the form shown in fig. 1. This loop body takes environmental information *Env* and a set of current navigational options *NOpts*. *Env* is sampled at each $NOpts_i$ to produce the same list of navigational options:

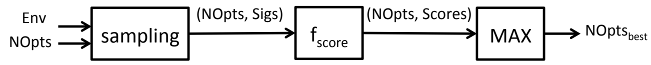

Fig. 1. Body of a generalised control loop mapping environmental information *Env* and current navigational options *NOpts* to the best navigational choice $NOpts_{best}$.

NOpts paired with corresponding navigational signals *Sigs*. The scoring function f_{score} combines the signals for each navigational option to produce a small set of numeric scores (signals) corresponding to that option. In this work, f_{score} is the subject of evolutionary search – all other components remain fixed. Finally a simple *MAX* function is used to determine the maximum score and the corresponding navigational option $NOpts_{best}$ is returned as the navigational target for this control iteration.

A number of approaches in the literature either conform or could be made to conform to this general schema [1], [2], [3], [4], [5], [7], [8], [9]. For example, in Yamauchi's navigational framework [2] *Env* can be considered to current pose and distance fields; *NOpts* is a set of unvisited frontier cells; and *Sigs* are the distances to the frontier cells. In contrast, for Knudson [5] *Env* consists of a field histogram and robot pose; *NOpts* are candidate bearings; and *Sigs* are clear-traversal distance, and bearing relative to goal. In this paper *NOpts* is a list of nearby candidate navigational poses; *Env* is a set of distance fields, current pose and a power-consumption model; and *Sigs* are relative-distance to goal, collision-hazard and peak power consumption.

Note that, for the loop body above to be effective, the signals in *Sig* must embed enough information to enable good decisions. For example, in our framework, the signal indicating relative-distance-to-goal for a navigation option is informed by a distance field which embeds all possible paths to the goal. In contrast, responding to a signal indicating goal direction, runs the risk of being stuck behind occluding obstacles.

Relating our evolutionary approach to others: while the use of GP for direct derivation of reactive control has a long history [10], [11], this work is the first to evolve the code structure of a scoring function for autonomous mapping. Neuro-evolutionary approaches to control also have a long history [12]. Of these, Knudson [5] appears most similar by evolving a neural net to assume the role of a scoring function to rank navigational directions. However, like all neuro-evolutionary approaches the evolved control is black-box and not open to inspection.

Evolutionary methods are also commonly applied to path planning [13], [14], [15]. This work differs from ours in that it involves dynamic creation of a path to a known target whereas our evolved function encodes the logic for immediate target choice.

III. EXPLORATION STRATEGY

In this section, we describe the robotic platform and mission objectives. These form the environmental constraints that guide the design of the control software and the evolution of the scoring function.

A. Robot Description

We consider a wheeled skid-steered robot of rectangular shape as in Fig. 2(a) to explore an unknown environment. This simulated non-holonomic robot is equipped with a 360^o laser-range finder to measure proximity data. Localisation and pose approximation are achieved by using a simulated global positioning unit (GPS) and an inertial management unit (IMU) readings. We are using direct data from the simulated sensor as our simulated robot kinematics are carefully derived from physics-based models. The use of direct localisation enables us to speed up our evolutionary framework without the cost of running the simultaneous localisation and mapping (SLAM) module. However, for a real robot implementation or later stages of evolution, a SLAM module can be integrated to improve pose and map accuracies.

The kinematic model of the robot is based on configuration transition equations [16] using translational velocity v_t (forward movement) and angular velocity v_t (rate-of-turn) as below:-

$$\dot{x} = rv_t cos\theta \qquad \dot{y} = rv_t sin\theta \qquad \dot{\theta} = \frac{r}{L} v_a \tag{1}$$

where $(\dot{x}, \dot{y})$ is change of robot pose in Cartesian coordinate and $\dot{\theta}$ is change of its bearing. L is the distance between two wheels, meanwhile r is the radius of robot wheels. Maximum velocities of the robot are set as $v_t = 2\frac{m}{s}$ and $v_a = 40\frac{deg}{s}$. The robot is moved using point-and-shoot motion where the robot first turns to the desired direction and then moves forward to the specified distance.

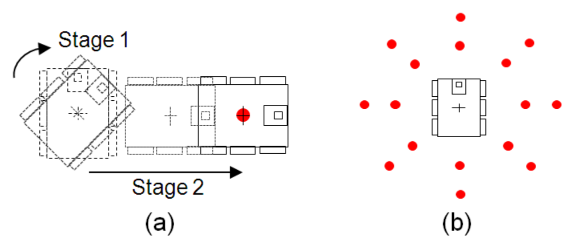

Fig. 2. (a) Robot's point-and-shoot motion. Stage 1 turns robot to its desired direction, stage 2 drives robot forward at its specified distance (b) Navigational Options

B. Exploration Strategy Framework

Our exploration strategy framework is adapted from our previous work in [7], [8] with few modifications to cater single robot exploration task. Furthermore, the exploration task has two objectives: maximising exploration and minimising power consumption; and one constraint: collision avoidance. Fig. 3 shows a block diagram of the exploration framework with respect to the robot described in III-A. In each control cycle t, current pose of the robot q_{robot}, direction and scan readings are acquired from the robot's

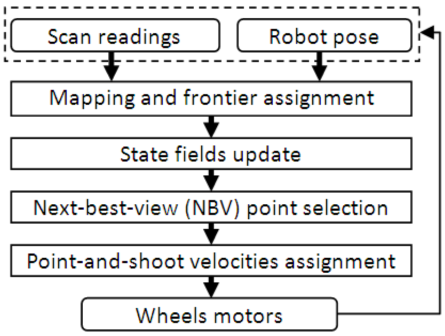

Fig. 3. Exploration strategy framework

sensors. These readings are used to build/update a two-dimensional occupancy grid **O**. Here, the resolution of a grid square of **O** is set to 10cm to obtain reasonable map accuracy. A Bayesian map update formula is used as in [17] to determine whether each grid square is empty, occupied or unexplored. Subsequently, we use a frontier-based strategy [2] to identify a long-range target locations. The closest frontier location to q_{robot} is selected as the target location: $q_{frontier}$. Note that, our approach differs in that the frontier is used as a proxy to inform the robot of the rough direction of the closest unknown area. Because the laser scan precedes the robot, $q_{frontier}$ is normally reassigned long before the robot reaches the old frontier.

Next, two auxiliary fields are derived from **O**: a static-hazard-distance field $\mathbf{F}_{sh}$ and a frontier-cost field $\mathbf{F}_{fc}$. $\mathbf{F}_{sh}$ is a global field that assigns each free cell in **O** with the distance to the nearest fixed obstacle, $dist_q \subset \mathbf{F}_{sh}$ [7]. $dist_q$ of an occupied or unexplored cell is set to zero. On the other hand, $\mathbf{F}_{fc}$ contains traversal costs from each cell in **O** to $q_{frontier}$, $cost_q \subset \mathbf{F}_{fc}$. Both fields are calculated using a deterministic variant of the value iteration algorithm [1]. An example of these fields is shown in fig. $4(b)$ and $4(c)$ corresponding to map in fig. $4(a)$.

The latest values of $\mathbf{F}_{sh}$ and $\mathbf{F}_{fc}$, combined with a power consumption model derived from the robot's kinematics form the environment *Env* which is sampled in the control loop in fig. 1. The components in *Env* are sampled at the navigational options q_i shown in fig. 2(b) to produce three signals for the scoring functions. These scalar signals are *target-strength*, embodying relative distance to $q_{frontier}$, extracted from $\mathbf{F}_{fc}$; *static hazard* embodying risk of collision, extracted from current robot pose and $\mathbf{F}_{sh}$; and, finally, *power consumption* is derived directly from kinematic models. These signals are combined using the scoring function f_{score} to assign a score to each navigational option. The best option, called next-best-view (NBV) becomes the short-range navigational target:q_{nbv}. The final process in the control cycle is to set velocities v_t and v_a to move the robot from q_{robot} to q_{nbv}. This low-level controller translates velocities to wheel rotation to execute the point-and-shoot motion to q_{nbv}.

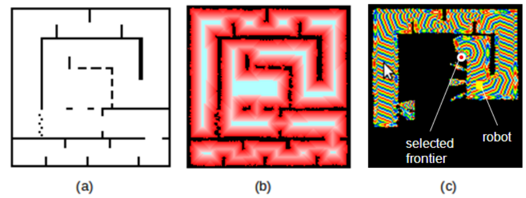

Fig. 4. (a) Exploration map, (b) Corresponding static hazard distance field (fully explored map), (c) Corresponding frontier cost field after certain robot movements

C. Selection of the Next Best View (NBV) Location

The sampling process to extract target-strength, static-hazard and power-consumption at each navigational option warrants further description. We briefly discuss each in turn.

The target-strength ts_i of each navigational option q_i measures the relative distance of q_i point to $q_{frontier}$, compared to the distance between q_{robot} and $q_{frontier}$. This can be calculated by sampling cells' cost value in $\mathbf{F}_{fc}$. The value of ts_i ranges between 0.0 and 1.0 where a value greater than 0.5 indicates a q_i closer to $q_{frontier}$ than q_{robot}. Meanwhile, a ts_i value less than 0.5 denotes a q_i as further from the desired target. Pseudocode to generate ts_i is shown in fig. 5.

1) Acquire traversal cost of moving from q_i to $q_{frontier}$.
$$cost_{q_i} \subset \mathbf{F}_{fc}$$
2) Acquire traversal cost of moving from q_{robot} to $q_{frontier}$.
$$cost_{q_r} \subset \mathbf{F}_{fc}$$
3) Calculate the Euclidean distance between q_{robot} and q_i, D.
4) Calculate target-strength value, ts_i.
$$ts_i = \frac{cost_{q_r} - cost_{q_i}}{2 * D} + 0.5 \in [0, 1]$$

Fig. 5. Pseudocode to generate target strength signal of a candidate NBV point

The static hazard, sh_i, of each q_i measures the risk of collision with static obstacles. sh_i is calculated by first checking for possible collisions when the robot moves from q_{robot} to q_i. If no collision detected, then, the normalised sh_i is calculated based on distance of q_i from the nearest obstacle, $dist_{q_i}$. Both processes use $\mathbf{F}_{sh}$ to extract distance information. Pseudocode to generate sh_i is shown in fig. 6.

Power consumed by the robot moving from q_{robot} to q_i can be estimated by summing the power used to turn the robot and power used to move the robot forward. This cumulative power can be calculated by using robot dynamic model taking into account v_t and v_a. We adopt models from [19] to calculate the power consumption signal, pw_i of every q_i. Pseudocode to generate pw_i is explained in fig. 7. The higher pw_i value the higher power consumed to drive robot to q_i and vice-versa.

D. The Scoring function

We apply a decision-theoretic approach to build a scoring function, f_{score} [1]. This approach allows several signals to

1) Apply a collision-detection algorithm. We use the Separating Axis Theorem technique [18].

$$sh_i = \begin{cases} 1 & \textit{if collide} \\ 0 & \textit{otherwise} \end{cases}$$

2) If $sh_i = 0$ (no collision), re-calculate sh_i in respect to $dist_{q_i}$ value.

$$sh_i = \frac{1}{mn - mx} dist_{q_i} - \left(\frac{mx}{mn - mx}\right) \in [0, 1]$$

where, mn is the lower bound value in F_{sh} before collision happens and mx is the upper bound value in F_{sh} before the robot moves to safe area.

Fig. 6. Pseudocode to generate static hazard signal of a candidate NBV point

1) Calculate power consumed to turn from q_{robot} direction, θ_r to q_i direction, θ_i.

$$p_{turn} = MR * |\theta_r - \theta_i|$$

2) Calculate power consumed to move robot forward from q_{robot} to q_i.

$$p_{fwd} = mass * a_t + FR * v_t$$

3) Calculate pw_i as

$$pw_i = \frac{p_{turn} + p_{fwd}}{powMax} \in [0, 1]$$

where MR is moment of resistance, FR is longitudinal resistance, mass is the robot mass in kg, a_t is the robot acceleration and powMax is the maximum estimated power consumption of the longest possible movement.

Fig. 7. Pseudocode to generate power consumption signal of a candidate NBV point

be combined in f_{score}. The general structure of our f_{score} for each q_i: f_{score_i} is as follows:-

$$f_{score_i} = f(sh_i, pw_i, ts_i) \tag{2}$$

Eq. 2 indicates that f_{score_i} is built from the aggregation of function of signals sh_i, pw_i and ts_i. In the next section, we describe our evolutionary mechanism to generate good forms for f_{score}. Note that, f_{score_i} returns a real value with a greater score indicating a more desirable q_i. f_{score_i} below or equal to 0.0 indicate that it is better for the robot to stay at q_{robot} than to move to q_i.

IV. EVOLUTION OF THE SCORING FUNCTION

In this section, we explain the evolutionary mechanism implemented in our exploration strategy. The objective is to find the best f_{score} we can in terms of exploration performance. We use Grammatical Evolution (GE) as the search framework. We briefly describe GE next followed by its use in deriving f_{score}.

A. Grammatical Evolution

Grammatical Evolution is an evolutionary framework that is used for the automatic generation of program code in arbitrary languages [20]. In GE, the framework is given a grammar for the problem domain in Backus-Naur Form (BNF). GE uses this grammar to map binary strings representing individuals to syntactically correct expressions in the grammar. The primary advantage of GE is that it gives the user the freedom to define grammars suited to their application without the risk of generating large numbers of syntactically incorrect individuals.

In this paper, we compose the BNF grammar to define particular elements in f_{score}. We describe experiments using three distinct grammars. The first grammar restricts GE to evolving only the constants of f_{score} with the rest of the structure fixed. The second grammar is more relaxed allowing different choices of unary functions with each term but restricting aggregation to simple multiplication of terms. The third grammar is most flexible, allowing terms and the aggregation operators to evolve freely. We detail these grammars next.

B. Experimental Grammar 1: Evolving Constants

In this grammar, constants are allowed to evolve but the rest of the grammar is a fixed product of linear terms. Several works in the literature use combinations of weighted linear terms [3], [4], [9] so this represents a benchmark of interest. Fig. 8 shows the BNF grammar for evolution of constants. The structure of f_{score} is fixed with multiplicative

<expr>::= $(1 - Linear(pw_i,$ <num_linr>$))$ *
$(1 - Linear(sh_i,$ <num_linr>$))$ *
$Linear(ts_i,$ <num_linr>$)$
<num_linr>::= <num_type1>, <num_type2>
<num_type1>::= <num10>.<num10><num10>
<num_type2>::= <sign><num2>.<num10><num10>
<num10>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<num2>::= 0 | 1
<sign>::= | -

Fig. 8. BNF Grammar for constants evolution

operators that combines linear functions of pw, sh and ts. The constants are num_type1 and num_type2 which represent value of a_1 and b_1, respectively, in a linear function as in eq. 3.

$$a_1 * x + b_1 \in [0, 1] \tag{3}$$

where x is a state signal value. a_1 has range between 0.00 to 9.99, while b_1 has value between -1.99 to 1.99.

C. Experimental Grammar 2: Evolving Functions

In certain problems, tuning the constants is not adequate to give a good solution. For example, a non-linear problem represents by a linear solution will often handle a problem ineffectively. Therefore, we propose an evolution of state signals' functions as well as its corresponding constants simultaneously. We extend the grammar in section IV-B to find the best function for each signal which can be chosen among the following function types: linear function, polynomial function or logistic function. Eq. 4 and 5 describe the polynomial and logistic functions, respectively, with its corresponding constants (a_2, b_2, a_3, b_3).

$$a_2 * (x^{b_2}) \in [0, 1] \tag{4}$$

$$\frac{1.0}{1.0 + \exp(-a_3 * (x - b_3))} \in [0, 1] \tag{5}$$

where the constants are ranged as follow: a_2 : 0.00 to 9.99, b_2 : 0.00 to 9.99, a_3 : 0.00 to 29.99 and b_3 : 0.00 to 0.99, Fig. 9 shows the extension of the BNF grammar from fig. 8 that is used to cater evolution of functions and constants.

```
<expr>::= (1− <pw_func>) * (1− <sh_func>) * <bs_func>;
<pw_func>::= Linear(pw, <num_linr>)          |
             Polynomial(pw, <num_poly>) |
             Logistic(pw, <num_lgtc>)
<sh_func>::= Linear(sh, <num_linr>)          |
             Polynomial(sh, <num_poly>) |
             Logistic(sh, <num_lgtc>)
<gs_func>::= Linear(gs, <num_linr>)          |
             Polynomial(gs, <num_poly>) |
             Logistic(gs, <num_lgtc>)
<num_poly>::= <num_type1>,<num_type1>
<num_lgtc>::= <num_type3>,<num_type4>
<num_type3>::= <num3><num10>.<num10><num10>
<num_type4>::= 0.<num10><num10>
<num3>::= 0 | 1 | 2
```

Fig. 9. BNF Grammar for functions evolution

D. Experimental Grammar 3: Evolving Structure

The grammar above still limits the structure of f_{score} to the product of terms dependent on ts, sh and pw. This fixed structure may rule out good design options. In this grammar we lift the restrictions on operators and allow the evolution of:

1) mathematical operators for aggregation between signals $(+, -, *, /)$.
2) order of signals in f_{score}.
3) the function for each signal.
4) corresponding constants for each signal term.

Fig. 10 presents a more general extension of the BNF grammar from from fig. 8 and 9 to accommodate structure evolution. Note that we include explicit options for different orderings and associativity of terms. These ensure that each signal is included once in any solution but gives maximum flexibility in how it contributes. With this general method, GE has a very much larger search space than solution in IV-B and IV-C. As a consequence, f_{score}'s candidate structure has larger diversity.

```
<expr>::=<pw_struc>; <sh_struc>; <bs_struc>; <scorefunc_struc>;
<pw_struc>::= pwf = <pw_func> | pwf = 1− <pw_func>
<sh_struc>::= shf = <sh_func> | shf = 1− <sh_func>
<gs_struc>::= gsf = <gs_func> | gsf = 1− <gs_func>
<scorefunc_struc>::= scorefunc = pwf<op>shf<op>gsf  |
                     scorefunc = (gsf<op>shf)<op>pwf   |
                     scorefunc = (gsf<op>pwf)<op>shf   |
                     scorefunc = (shf<op>gsf)<op>pwf   |
                     scorefunc = (shf<op>pwf)<op>gsf   |
                     scorefunc = (pwf<op>gsf)<op>shf   |
                     scorefunc = (pwf<op>shf)<op>gsf
<op>::= +| − | ∗ |/
```

Fig. 10. BNF Grammar for structure evolution

V. EXPERIMENTAL SETUP

Three different experiments were designed to evaluate the proposed evolution of f_{score}. The first experiment (CONST_EVO) uses the BNF grammar in fig. 8 to tune constants of linearised f_{score}. Meanwhile, the second (FUNC_EVO) and third (STRUC_EVO) experiments implement the BNF grammars as in fig. 9 and 10, respectively.

For all experiments, any candidate function, f_{cand} produced by the BNF grammar is scored by embedding f_{cand} in a high-fidelity simulation platform. The evaluative process for each f_{cand} is:

1) *embedding*: f_{cand} is substituted into the C++ source code as f_{score} of the exploration strategy and then code re-compilation is performed.
2) *simulation*: the exploration strategy with the embedded f_{cand} is run in a simulated environment. We use a map emulates the office-like environment as in fig. 4(a) to measure robot performance. Stage, a software development platform [21] is utilised to perform the simulation. Simulation time-frame and maximum power usage are set. The robot must explore the map within the time-frame. The simulation is terminated whenever the following conditions occur first: i) runtime exceeds time-frame (we set 180 seconds), ii) collision, iii) robot in stall position, iv) robot consumes power more that the maximum power usage, or v) robot explores the map completely. With simulation time sped up between 20-50 times faster, we are able to improve evolution time significantly.
3) *evaluation*: Fitness for every f_{cand} is taken using the equation 6:

$$fitness = \left(\mathit{area} + \frac{1}{1 + \mathit{power}}\right) * \left(\frac{1}{1 + \mathit{coll}}\right) \quad (6)$$

where *area* is the total area explored, *power* is the total power consumed and *coll* is the number of collisions with obstacles. This fitness function is created to reflect the multi-objective exploration task as described in section III-B. It indicates that a f_{cand} with larger area of exploration with relatively low power usage is rewarded with higher score. Collisions penalise the fitness.

Experiments were conducted on an IBM HS22 server with an 3.47GHz eight core Intel Xeon X5677 CPU and 48GB of memory. We ran each experiment 15 times. The longest evolutionary runs would consume 2 days of CPU time. In GE, we set population to 100 and generation to 100, 200 and 300 for CONST_EVO, FUNC_EVO and STRUC_EVO, respectively. Different numbers of generations were used to allow enough time for convergence in each experiment[1]. We use a standard one-point crossover and point-mutation.

VI. RESULTS AND DISCUSSION

The experimental results for the above evolution are considered in this section. Fig. 11 shows the statistical data of all runs represented in box plots (boxes span the 25th to 75th percentiles). The first observation found that f_{score} from CONST_EVO produces the lowest performance

[1]Due to its smaller search space CONST_EVO typically converged in only 50 generations.

(median fitness:63) compared to f_{score} from FUNC_EVO (median fitness:188) and STRUC_EVO (median fitness:186). This reveals that f_{score} in linear form is not adequate to produce the best strategy in this framework. In contrast, results from FUNC_EVO and STRUC_EVO present a significant improvement to the exploration performance. Both experiments prove that good solution can be achieved with the right selection of f_{score} structure. STRUC_EVO has advantages over FUNC_EVO in that the complete structure of f_{score} is designed automatically. In contrast, FUNC_EVO requires designers to manually define the aggregation method. Equations 7 to 9 present the best found f_{score}s from the three experiments.

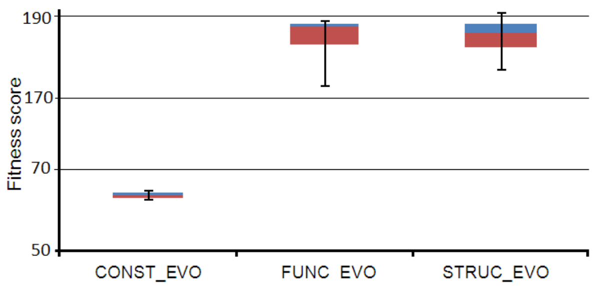

Fig. 11. Boxplot comparison of f_{score} evolutions

CONST_EVO:

$$\begin{aligned} s &= 4.93 * sh - 1.93 \quad \in [0,1] \\ p &= 0.90 * pw - 0.02 \quad \in [0,1] \\ t &= 0.54 * ts + 0.11 \quad \in [0,1] \\ f_{score} &= (1-s)*(1-p)*t \end{aligned} \tag{7}$$

FUNC_EVO:

$$\begin{aligned} s &= 7.6\,(sh^{5.61}) \quad \in [0,1] \\ p &= 0.96\,(pw^{9.62}) \quad \in [0,1] \\ t &= \frac{1}{1+exp(-26.9*(ts-0.62))} \quad \in [0,1] \\ f_{score} &= (1-s)*(1-p)*t \end{aligned} \tag{8}$$

STRUC_EVO:

$$\begin{aligned} s &= 2.92\,(sh^{4.67}) \quad \in [0,1] \\ p &= \frac{1}{1+exp(-16.27*(pw-0.06))} \quad \in [0,1] \\ t &= \frac{1}{1+exp(-20.75*(ts-0.55))} \quad \in [0,1] \\ f_{score} &= (1-s)/p*t \end{aligned} \tag{9}$$

A. Validation

Once the optimised f_{score} has been found, the robot can be used to explore any new unknown environment that has almost the same features as the map used in the evolution process. In order to validate the performance of the findings, we test f_{score} as in eq. 7 to 9 on another simulated office-like environments as in fig. 12. We run the simulation until the whole maps are covered by the robot. After 10 runs of each function, we found the average performance of the robot on both maps as per table I. The table shows that the robot explores the maps completely with f_{score} produced by FUNC_EVO and STRUC_EVO. However, f_{score} produced by CONST_EVO unable to complete the exploration and stop after it gets stalled. Comparing power consumption, FUNC_EVO uses less power than CONST_EVO and STRUC_EVO as the evidence shows that the total path length the robot moves is shorter than the others. In overall, we can conclude that f_{score} of FUNC_EVO and STRUC_EVO outperforms CONST_EVO proving that the ideal relationship between signals and scores, in our setup, is unlikely to be expressed using linear terms.

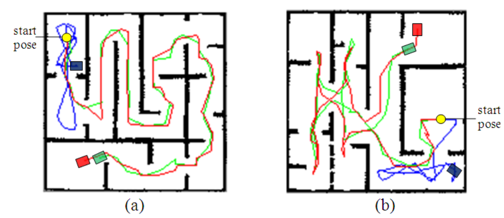

Fig. 12. Test maps showing footprints of the robot using three different f_{score}: red line for STRUC_EVO, green line for FUNC_EVO, and blue line for CONST_EVO (a) test map 1 at 240 sec (b) test map 2 at 234 sec

TABLE I

PERFORMANCE COMPARISON OF EVALUATIVE FUNCTIONS. PERFORMANCE PARAMETERS: A-EXPLORED AREA, T-EXPLORATION TIME, P-CONSUMED POWER AND TP-TOTAL PATH LENGTH [TESTMAP1/TESTMAP2]

f_{score}	A(m^2)	T(sec)	P(kW)	TP(m)
CONST_EVO	51/145	∞/∞	4.02/5.47	64.02/94.52
FUNC_EVO	225/225	217/236	3.52/3.77	58.72/62.38
STRUC_EVO	225/225	240/234	3.83/3.97	62.74/65.55

B. Discussion

From the experimental results, we have shown the influence of the selection of f_{score} to the exploration performance. By using GE to search for the best structure of f_{score}, we manage to improve the exploration task significantly. Using f_{score} from FUNC_EVO and STRUC_EVO as the choices, area coverage is significantly improved while maintaining reasonable power consumption.

To further investigate what is happening in these experiments, we represent f_{score} of CONST_EVO, FUNC_EVO and STRUC_EVO graphically. We reduce the dimension of f_{score} graph by keeping pw constant. In this case, we can analyse the relationship between ts and sh and their influence to the navigational choice. In general, signal-to-signal relationship analysis can be performed by assigning other signals with fixed values. Fig. 13 shows the ranking surfaces of f_{score} for CONST_EVO, FUNC_EVO and STRUC_EVO at $pw = 0.1$.

From the figure, the highest score is at 'peak' point indicating a point with the highest ts and the lowest sh as the best point to navigate. Meanwhile, the lowest score with zero value is in 'valley' (region in black) where those points are unlikely to be chosen as the navigational point. In comparison, we can see that STRUC_EVO and FUNC_EVO are able to be more aggressive in exploration than CONST_EVO. To illustrate, navigation choice A shown in each surface, is a high-pay-off(ts=0.9)/high-hazard(sh=0.6, but considerably safe) navigational choice. In CONST_EVO choice A has a very low ranking - it will almost never be chosen while in STRUC_EVO and FUNC_EVO it ranks very highly. Conversely, choice B which is low-pay-off(ts=0.5)/low-hazard(sh=0.4) is relatively attractive in CONST_EVO but ranked very poorly in STRUCT_EVO and FUNC_EVO.

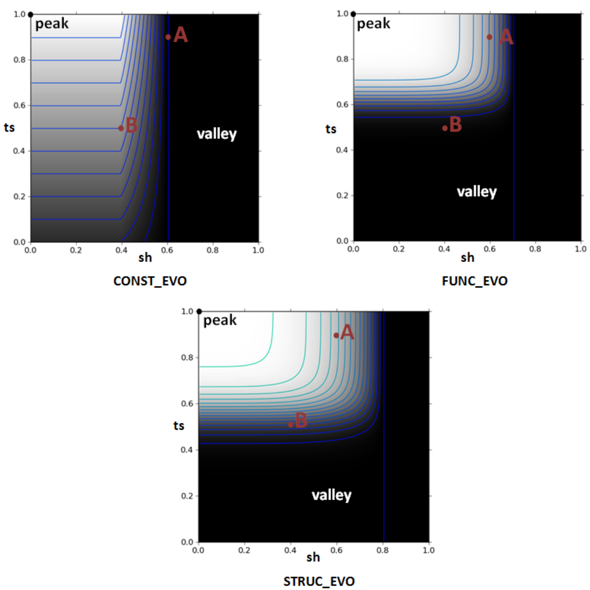

Fig. 13. Decision making surfaces

Therefore, we can infer that the ranking surfaces of STRUC_EVO and FUNC_EVO yield better navigational decision making compared to CONST_EVO. The most persistent features of the ranking surfaces of STRUC_EVO and FUNC_EVO are being the sharply defined low plain for sh above 0.8, the low plateau for ts below 0.5 and the gentle rise for ts values above 0.5. This echelon structure, and its boundaries, seem to characterise what is required of a good f_{score} in our particular setup.

VII. CONCLUSIONS AND FUTURE WORKS

To achieve optimal robotic exploration performance in an unknown environment, one must adopt an optimal decision making function for the given exploration strategy framework. In this paper, we have introduced the mechanism to design the structure of a decision making function automatically using Grammatical Evolution (GE). Experimental results had shown that GE is able to search for good function structures in a very large search space and improve the exploration performance significantly.

For future works, we would like to introduce more state signals to take account of more complex environments containing moving obstacles and multiple robots. We aim to use multi-stage evolution for the evolution of structure and then constants. We would like to evolve individuals in a larger variety of environments with differing levels of noise to provide insight into the impacts these have on the scoring function. Finally, we would like to use diverse individuals, evolved under different conditions as a pool to underpin real-time adaptation on real robotic platforms.

REFERENCES

[1] C. Stachniss, Decision-Theoretic exploration using coverage maps, Robotic Mapping and Exploration, 2009.

[2] B. Yamauchi, Frontier-based exploration using multiple robots, in Proc. of the 2nd Int. Conf. on Autonomous Agents, USA, 1998, pp. 47-53.

[3] W. Burgard, M. Moors, C. Stachniss, and F.E. Schneider, Coordinated multi-robot exploration, IEEE Trans. Robotics, vol. 21, no. 3, pp. 376-386, 2005.

[4] A. Mobarhani, S. Nazari, A.H. Tamjidi, and H.D. Taghirad, Histogram based frontier exploration, in Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, USA, 2011, pp. 1128-1133.

[5] M. Knudson, and T. Kagan, Adaptive navigation for autonomous robots, Journal of Robotics and Autonomous Systems, vol. 59, no. 6, pp. 410-420, 2011.

[6] I. Harvey, Artificial evolution and real robots, Artificial Life and Robotics 1(1),pp. 35-38, 1997.

[7] M.F. Ibrahim, and B. Alexander, Evolving a path planner for a multi-robot exploration system using grammatical evolution, in Proc. of the 7th Int. Conf. on Intelligent Sensors, Sensor Networks and Information Processing, Australia, 2011, pp. 590-595.

[8] M.F. Ibrahim, and B. Alexander, Designing a navigational control system of an autonomous robot for multi-requirements planetary navigation using evolutionary algorithms approaches, in Proc. of the 12th Australian Space Science Conf., Australia, 2012, pp. 223-234.

[9] N. Basilico, and F. Amigoni, Defining effective exploration strategies for search and rescue applications with multi-criteria decision making, in Proc. of IEEE Int. Conf. on Robotics and Automation, 2011, pp. 4260-4265.

[10] P. Nordin and W. Banzhaf, An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming, Adaptive Behaviour, 5(2), pp. 107–140, 1997.

[11] M. Mataric and D. Cliff, Challenges in evolving controllers for physical robots, Robotics and Autonomous Systems, 19(1), pp. 67–83, 1996.

[12] J. Meyer, P. Husbands and I. Harvey, Evolutionary robotics: A survey of applications and problems, Evolutionary Robotics, pp. 1-21, Springer Berlin/Heidelberg, 1998.

[13] H. Mahjoubi, F. Bahrami and C. Lucas, Path planning in an environment with static and dynamic obstacles using Genetic Algorithm: A simplified search space approach, in Proc. IEEE Congress on Evolutionary Computation, 2006, Canada, pp. 2483-2489.

[14] J. Chakraborty, A. Konar, U.K. Chakraborty and L.C. Jain, Distributed cooperative multi-robot path planning using differential evolution, in Proc. IEEE Congress on Evolutionary Computation, 2008, pp. 718-725.

[15] M. Naderan-Tahan, and M.T. Manzuri-Shalmani, Planning a robust path for mobile robots in dynamic environment, in Proc. 14th Int. CSI Computer Conference, 2009, pp. 470-476.

[16] S.M. LaValle, Planning algorithms, Cambridge University Press, Cambridge, U.K., 2006.

[17] L. Matthies, and A. Elfes, Integration of sonar and stereo range data using a grid-based representation, in Proc. of 1988 IEEE Int. Conf. on Robotics and Automation, 1988.

[18] D. Eberly,Intersection of Convex Objects: The Method of Separating Axes, http://www.geometrictools.com/, 1998.

[19] O. Chuy, E.G. Collins, Wei Yu and C. Ordonez, Power modeling of a skid steered wheeled robotic ground vehicle, in Proc. of IEEE Int. Conf. on Robotics and Automation, 2009, pp. 4118-4123.

[20] M. ONeill and C. Ryan, Grammatical Evolution, IEEE Transactions on Evolutionary Computation, vol. 5, no. 4, pp. 349-358, 2001.

[21] R. Vaughan, Massively multi-robot simulation in stage, Swarm Intelligence Journal, vol. 2, pp. 189-208, 2008.